DOI:
10.1145/3134600.3134632

Document Version
Peer reviewed version

Ex-Ray: Detection of History-Leaking Browser Extensions

Michael Weissbacher
Northeastern University
mw@ccs.neu.edu

Enrico Mariconti
University College London
e.mariconti@cs.ucl.ac.uk

Guillermo Suarez-Tangil
University College London
guillermo.suarez-tangil@ucl.ac.uk

Gianluca Stringhini
University College London
g.stringhini@ucl.ac.uk

William Robertson
Northeastern University
wkr@ccs.neu.edu

Engin Kirda
Northeastern University
ek@ccs.neu.edu

ABSTRACT

Web browsers have become the predominant means for developing and deploying applications, and thus they often handle sensitive data such as social interactions or financial credentials and information. As a consequence, defensive measures such as TLS, the Same-Origin Policy (SOP), and Content Security Policy (CSP) are critical for ensuring that sensitive data remains in trusted hands.

Browser extensions, while a useful mechanism for allowing third-party extensions to core browser functionality, pose a security risk in this regard since they have access to privileged browser APIs that are not necessarily restricted by the SOP or CSP. Because of this, they have become a major vector for introducing malicious code into the browser. Prior work has led to improved security models for isolating and sandboxing extensions, as well as techniques for identifying potentially malicious extensions. The area of privacy-violating browser extensions has so far been covered by manual analysis and systems performing search on specific text on network traffic. However, comprehensive content-agnostic systems for identifying tracking behavior at the network level are an area that has not yet received significant attention.

In this paper, we present a dynamic technique for identifying privacy-violating extensions in Web browsers that relies solely on observations of the network traffic patterns generated by browser extensions. We then present Ex-Ray, a prototype implementation of this technique for the Chrome Web browser, and use it to evaluate all extensions from the Chrome store with more than 1,000 installations (10,691 in total). Our evaluation finds new types of tracking behavior not covered by state of the art systems. Finally, we discuss potential browser improvements to prevent abuse by future user-tracking extensions.

1 INTRODUCTION

The browser has become the primary interface for interactions with the Internet, from writing emails, to listening to music, to online banking. The shift of applications from the desktop to the Web has made the browser the de-facto operating system. To augment this experience browsers offer a powerful interface to access and modify websites. Among the available functionalities, extensions can modify HTTP requests and responses, inject content into websites, or execute programs as a background activity. This allows for extensions that manage passwords, remove ads, or store bookmarks in the cloud.

The downside of this powerful interface is that malicious actions at the extension level can lead to problems across all online activities for a user. Extensions can be considered as the "most dangerous code in the browser" [9]. Previous research found extensions to inject or replace ads [1, 10, 29], causing monetary damage to content creators and, in turn, consumers. To detect privacy-invasive extensions, previous work used dynamic taint analysis to find spyware in Internet Explorer Browser Helper Objects (BHOs) [5]. With previous research in mind, browser vendors can work to restrict malicious extensions.

Google Chrome is considered the state of the art in secure browsing. Chrome extensions can only be installed through a centralized store, and before being admitted they have to pass a review process. Similar to Android apps, Chrome extensions can request permissions to perform certain activities, and users can use this information to decide whether they want to install the extension or not. Furthermore, if an extension is considered malicious after admission to the store, it can be remotely removed from clients. With all these security features in mind, privacy in Chrome extensions is still an open issue.

This work aims to understand to what extent browser extensions violate user privacy expectations. In preliminary experiments, we found suspicious activity in popular browser extensions and confirmed that data is not only leaked, but furthermore is processed by third parties. By presenting unique URLs to multiple extensions, we were able to link incoming connections on a honeypot to the particular extension responsible for leaking user data.

Inspired by these findings, we introduce Ex-Ray, a system that can automatically detect history-stealing browser extensions without depending on the specific protocol used or leaking methodology. Our automated approach is based on analyzing the network traffic generated by dynamically exercising unmodified extensions. Extensions under test are executed within an instrumented browser multiple times, and all network traffic generated during execution is recorded. We decided to focus on the network activity generated by browser extensions because, while their code and logic can change, they ultimately need to send the acquired information to their controller, and this will be observable from network traffic. Thus, our approach builds on a fundamental invariant of tracking and user privacy violation. Furthermore, long term studies of malware have highlighted network activity as a particularly effective

ACSAC 2017, San Juan, PR, USA

DOI: 10.1145/3134600.3134632

medium for detecting malicious activity [12]. We model features that are intrinsic to the network traffic generated by trackers to distinguish malicious from benign traffic. We create complementary detection systems in unsupervised and supervised fashion, and a triage system which can classify the likelihood of a leak, easing the burden on security analysts to identify misbehaving extensions. After identifying a set of extensions that leak private information by looking at their network traffic, we develop a complementary component that can infer if an extension is leaking sensitive information by looking at the API calls that it invokes.

Ex-Ray automatically flagged 212 potential trackers in the top 10,691 extensions with a false detection rate of 0.27%. Our system has found two tracking extensions which were not detected by previous systems because they were leaking information using a different channel than what was expected by those tools. More precisely, one extension made use of strong encryption to obfuscate its behavior, and the other used WebSockets to exfiltrate user information as opposed to HTTP(S).

In summary, the contributions of this work are as follows:

- We developed the first unsupervised system to detect history-stealing browser extensions based on network traffic alone that is also robust against obfuscation.
- We quantify the magnitude of user data leakage and introduce a scoring system that is used to triage extensions. Prioritized extensions are manually vetted and the resulting labeled dataset is made available to the research community.[1]
- We created a machine learning approach to classify extensions that we use on API call traces generated by an instrumented browser. This approach reaches 96.43% F-Measure value and the recall value is constantly over 99%.

2 MOTIVATION

This work focuses on tracking data collected from browsing behavior that is sent to third parties. As opposed to previous work on history-leaking browser extensions [24], we aim for a system that will detect leaks regardless of how they are transmitted or collected. We target tracking either through background scripts or modification to pages. The main difference between these two approaches is that in the Web such trackers are only present on websites that opt-in to use them. From a user's perspective, tools that remove these trackers are available and well understood. Conversely, tracking in extensions can cover all websites a user visits, and there is no opt-in mechanism. Furthermore, no tools are readily available that would warn a user of such behavior or block it.

Transferring the current host or URL can be a necessary part of the functionality of an extension – for example, to check against an online blacklist such as an adult content filter. However, we found that often extensions also transfer URLs if no such checking is necessary, or could be expected by the extension's description, exposing all browsing habits of a user and creating a breach of privacy. Furthermore, the specification of such functionality is often buried deep in an extension's description, if present at all. Web users are concerned about how their privacy is impacted [2, 15], but are often unaware of what a privacy policy is.[2]

To provide additional context behind this sort of systemic privacy-violating behavior on the part of browser extensions, we present a detailed case study on a large actor in the history data collection market in Appendix A. In it, we demonstrate how a single library was tied to browsing data exfiltration in 42 extensions with over 8 million installations. The extensions were deleted from the Chrome Web Store within 24 hours of reporting.

2.1 HTTP URL Honeypot

To gain insight into the environment in which trackers operate, we configured a honeypot. To test whether leaked URLs are accessed after being received by trackers, we exercised extensions with domain names into which we encode their unique extension ID. While executing in our container, extensions only interact with local Web and DNS servers. However, we operate a Web server on the public Internet to monitor client connections for such URLs. As these domains are used uniquely for our experiments, HTTP connections indicate leaks linkable to extensions. The connection and execution times are displayed in Figure 1, and discussed in more detail in Section 4. The confirmation that trackers are acting on leaked data motivated further steps in this work. After excluding VPN and proxy extensions, we received incoming connections from 38 extensions out of all Chrome extensions with more than 1,000 users.

2.2 Types of Trackers

Chrome offers a powerful interface to extensions, and while it can be used to enhance the browsing experience, it can also be misused to violate user privacy. There are multiple ways to collect and exfiltrate browsing history.

Much like trackers that are added to Web pages by their authors, extensions can leak history by adding trackers to the body of Web pages. An example of third-party tracking is the Facebook "Like" button. These can be blocked by extensions such as Ghostery. A more robust solution is sending collected history data via requests of extension background scripts. Such requests are not subject to interception by other extensions, and cannot be blocked as tracker objects. Compared to tracking via inserting trackers into pages, better coverage can be achieved.

To acquire browsing data, extensions can intercept requests made by websites via the `chrome.webRequest` API, or poll tabs for the URL using `chrome.tabs`. For past browsing behavior, the `chrome.history` API can be used. Diverse options to collect data render finding a unified way to identify tracking extensions challenging.

2.3 Threat Model

Based on our honeypot results, we assume the following attacker model. In our scenario the attacker is the owner of, or someone who controls the content of, browser extensions. We assume many users will install these extensions with a cursory reading of the extension's description. While permissions can restrict the behavior of browser extensions, capturing and exfiltrating history can

[1] https://github.com/mweissbacher/exray-data

[2] http://www.pewresearch.org/fact-tank/2014/12/04/half-of-americans-dont-know-what-a-privacy-policy-is/

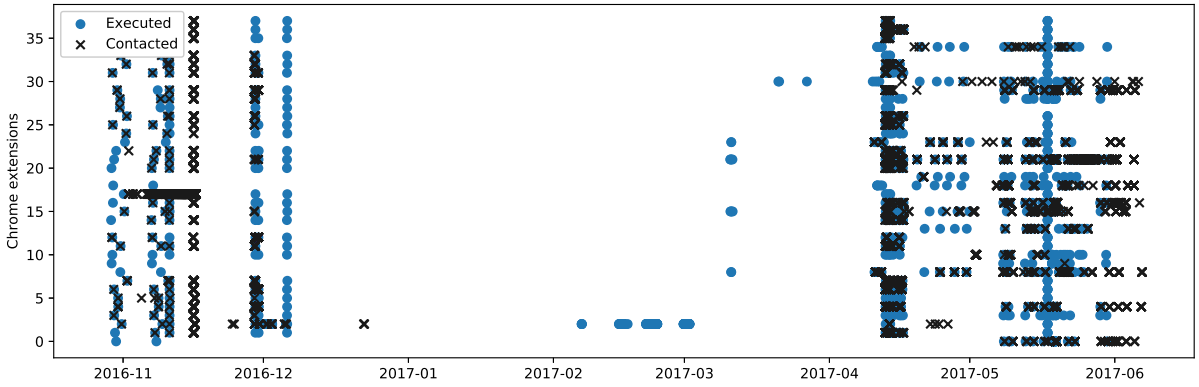


Figure 1: Extension execution with unique URLs compared to incoming connections to those URLs from the public Internet. These connections confirm that leaked browsing history is used by the receivers, often immediately upon execution.

be performed with modest permissions that would not raise suspicion. For instance, the browsing history permission is categorized as *low alert* by Google.

The goal of our attacker is to indiscriminately capture URLs of pages visited by the user while the extension is executed. Furthermore, we assume the adversary collects data with the purpose of analysis or monetization. As the value of traffic patterns decreases over time, we assume the attacker to be inclined to leak sooner rather than later, which seems to be confirmed by our honeypot experiments. A successful attacker would decrease the user's privacy as compared to using a browser without the extension in question.

We exclude from our threat model extensions that openly require the sharing of browsing history as part of their functionality, such as VPNs. Also, we consider leaks purposeful and supposedly accidental as equal, as we cannot reason well about developer intent. As detecting and hiding malicious behavior is an arms race, we prefer to be conservative and assume the attacker could escalate the sophistication of their evasion techniques in the future.

3 OUR APPROACH

In this section we describe the design of the approach underlying Ex-Ray. To identify privacy-violating extensions, we exercise them in multiple stages, varying the amount of private data supplied to the browser, and in turn to the extension under test. Based on the type of extension, the traffic usage can change depending on the number of visited sites. However, the underlying assumption is that benign extension traffic should not be influenced by the size of the browsing history.

3.1 Overview

A high level overview of Ex-Ray is shown in Figure 2. The three main components of the system are summarized as follows:

1) **Unsupervised learning**: We use counterfactual analysis to detect history-stealing extensions based on network traffic. This component is fully unsupervised and, by definition, prone to misinterpretations.
2) **Triage-based analysis**: We manually vet the output of our unsupervised system, i.e., we verify which extensions are factually leaking and which are not. As the manual verification is costly, we rely on a scoring system that ranks extensions based on how likely they are to be leaking information to aid the process.
3) **Supervised learning**: We systematize the identification of suspicious extensions using supervised learning over the resulting labeled dataset. This component takes into account the behavior of the extension and builds a model that detects history leaks (i.e., it looks at the API calls made by the browser extension when executed).

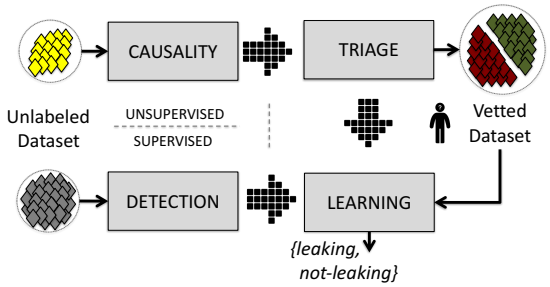
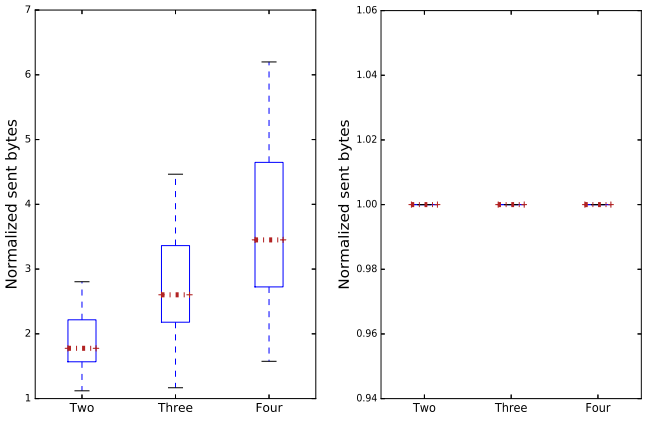


Figure 2: Ex-Ray architectural overview. A classification system combines unsupervised and supervised methods. After triaging unsupervised results, a vetted dataset is used to classify extensions based on n-grams of API calls.

We see different types of tracking used in browser extensions. Some intercept requests and issue additional requests to trackers. Others transfer aggregate data periodically, while still others insert trackers into every visited page. An integral part of all trackers is transferring data to an external server—simply put, this crucial step is what enables trackers to track.

Our work focuses on indiscriminate tracking across all pages. To track, a history item (h_i) generated by the browser will be reported either in isolation or in aggregate. In either case, the size of history items affects network behavior. We argue that network data generated by an effective tracker, independently of protocol



(a) Tracking extension. **(b) Benign extensions.**

Figure 3: Comparison in change of traffic between extensions leaking history and benign ones. With increasing history, each bar displays the change of sent data. For extensions that leak, sent data projects an ascending slope as a function of size of history.

and whether plain, encrypted, or otherwise obfuscated has to grow as a function of history.

We execute extensions in multiple stages increasing the amount of private information. Each h_i should contain less information than the following stage, $h_i < h_{i+1}$. We increase the size of h_i in each stage, extending the length of the testing URLs. For example, `example.com/example/index.html` in stage 0, and `example.com/example/<500characters>/index.html` in stage 10. The expected growth in traffic is h_Δ. This intuition is confirmed from Figures 3a and 3b where the boxplots clearly show that trackers usually send more data when there is more history to leak while the amount of data is constant across the different stages for benign extensions.

For deterministic tracking, the traffic deltas of adjacent measurements should project an ascending slope. However, the browser history may be sent compressed in order to send as few bytes as possible and avoid the leak being visible as plain text in the payload. This operation would reduce the number of bytes sent while retaining the same amount of information (entropy). Per information theory, message entropy has an upper bound that cannot be exceeded. As consequence, the size of compressed messages has a lower bound as a function of the message entropy. For our experiments, we used compression tools (bzip2, 7zip, xz) to establish a practical lower bound of sent data for each stage as 289 Bytes, 6.9 KB, 14 KB and 30 KB.

Extensions that use trackers establish connections with each execution. Consequently, any group of hosts that results in less measurements than the number of executions will not be considered for further analysis. Examples of hosts that extensions only connect to occasionally are ads.

3.2 Network Counterfactual Analysis

The goal of this phase is to model the way in which modifications to the browsing history influence observed network traffic. Figure 3 shows that there is a monotonic increase in sent traffic between successive stages of privacy-violating extension. Extensions that, on the contrary, are privacy-respecting show no significant difference. An observation we made during the analysis of traffic behavior is that privacy-violating extensions might exhibit non-leaking behavior when connecting to certain domains. Thus, it is important to consider individual flows when building our model. Additionally, we observed that variations exhibited by privacy-violating extensions are well-fit by linear regression.

Thus, we use linear regression on each set of flows to estimate the optimal set of parameters that support the identification of history-leaking extensions. We aim to establish a causality relation between two variables: (i) the amount of raw data sent through the network, and (ii) the amount of history leaked to a given domain. For this, we rely on the counterfactual analysis model by Lewis [13], where:

> The model establishes that, in a fully controlled environment, if we have tests in which we change only one input variable, and we observe a change in the output, then the variable and the output are linked by a relation of *causality*.

In our case, the *input* variable is the amount of history, the *output* is the number of bytes sent in the different flows, and the *tests* are run with both malware and benign samples. Our framework allows us to evaluate this relationship by means of different statistical tests, such as Bayesian inference. This is ideal for situations were there is no deterministic relationship between the variables, such as in targeted advertisement tracking. Although our framework is designed to model these scenarios, in practice, we observed that leaking extensions behave in a deterministic fashion.

In order to systematically identify the conditions under which the *causality* link is established, we run three steps. The first step is performed before applying linear regression, while the second and third steps are based on the linear regression parameters.

(1) **Minimum Intercept**. While the extension might communicate to a domain in all given stages, the content transmitted may not contain a privacy leak. This step verifies whether the amount of data sent exceeds a certain threshold. This threshold is set based on the size of the history compressed as described in Section 3.1.
(2) **Minimum Slope**. In this work we are primarily interested in extensions that actively track users. This type of extensions is expected to leak as much history data as possible from the user. This implies that the relationship between stages is expected to be linear and have a constant variance, modulo any sort of attempt at obfuscation. Based on this, we set a threshold to the slope in order to exclude all those extensions that do not fully meet these two criteria.
(3) **Level of Confidence**. Depending on the extension, the fitted regression model might not always be strictly linear. We can choose to apply certain bounds (lower, upper, or both) from a fitted model to adjust the precision of the output. Choosing bounds that are very close to the fitted

model will give a higher level of confidence in the decision. On the contrary, a very relaxed model will capture boundary cases at the cost of introducing false positives.

We define the term *flagging policy* as the set of parameters used for these checks. A *strict policy* is a policy in which parameters select a restricted area and flag less flows than a *relaxed policy* which flags many more flows as suspected of leaking browsing history.

The notion of confidence described above and the use of the different policies is precisely what motivates our triage system described next.

3.3 Extension Triage

After determining which extensions could potentially be leaking history in an unsupervised way, we manually vet those results. The goal of this phase is to design a score that quantifies history leakage and prioritize the manual analysis. For that, we first define a function L that estimates the number of URLs leaked between two controlled experiments such that

$$\mathsf{L}(s_i, s_j) = \frac{|s_j| - |s_i|}{\tau}. \quad (1)$$

Here, $|s_j|$ and $|s_i|$ are the number of bytes sent to a given domain in stages i and j respectively, while τ is a threshold that estimates the expected growth h_δ between i and j. This threshold is based on the size of the URLs used in the experiment described in Section 3.1. We abuse notation for the sake of simplicity and denote s when we refer to a transition between two consecutive stages with increasing exercised browsing history.

Given an extension x, we then obtain a score by multiplying the number of URLs leaked between two consecutive stages s:

$$\text{score}(x) = \prod_s e^{\mathsf{L}(s)}. \quad (2)$$

Note that we aim at obtaining a rough understanding of which extensions could be leaking URLs. Thus, we are mainly interested in high values of $\mathsf{L}(s)$. We use an exponential function of $\mathsf{L}(s)$ to prioritize extensions that are leaking in all stages but also to find a trade-off between extensions that only show a leak in some of the stages. Positive and large values will then output a high score.

Negative values of $\mathsf{L}(s)$ mean that the total amount of data sent in the earliest stage is higher than the amount sent in the latest. Intuitively this means that data sent in each stage does not depend on the size of URLs in the browser history. Our score then treats these cases as an exponential decay function, giving less weight to those stages and outputting values closer to $\text{score}(x) \simeq 0$. Likewise, when the amount of data exchanged between stages is exactly the same (i.e., $\mathsf{L}(s) \simeq 0$), the scoring function will then output $\text{score}(x) \simeq \prod e^0 \simeq 1$. One could obtain a probability of the likelihood of a leak by scaling the score to the interval $[0, 1]$. However, as our system currently aims only at prioritizing extensions, giving a rough notion of risk (without scaling the output) suffices. Thus, we consider the following thresholds as a general rule of thumb when triaging extensions:

$$\text{leak}(x) = \begin{cases} \texttt{not-leaking} & \text{if } score(x) \leq 1 \\ \texttt{possibly-leaking} & \text{if } 1 < score(x) \leq 100 \\ \texttt{likely-leaking} & \text{otherwise.} \end{cases} \quad (3)$$

Very large values of $\text{score}(x)$ show a high confidence that the extension is aggressively tracking users.

It is important to highlight that the triage stage is optional. An ideal setting with endless human resources would obviate the need to prioritize extensions and, thus, render the triage stage merely informative. In practice, extension markets are very large and human workers tend to be constrained time-wise, which can be a bottleneck for the verification process. In this case, having a triage system would be valuable. For the purposes of this paper, when labeling the outputs given by the unsupervised module, we have invested human effort in manually vetting extensions that are primarily ranked as `likely-leaking` (for positive samples) and `not-leaking` (for negative samples). We also look at a fraction of `possibly-leaking` extensions in a best-effort fashion.

3.4 History Leakage Detection

The last component of our system aims at systematizing the identification of unwanted extensions from a behavioral point of view. For that, we instrument the browser to monitor dynamically relevant behaviors of the extension during runtime. This component operates in a fully supervised manner and is composed of the following two phases:

- **Learning**: The system is trained using the dataset labeled in the previous phase, building a model of the most discriminatory runtime behaviors.
- **Detection**: The system is deployed to detect previously unknown privacy-violating extensions.

Predictions can be then used (i) to obtain a better understanding of how extensions (mis)use the user's private information; and, (ii) to discover previously unknown privacy-violating extensions than can then be analyzed by the triage component. As a result, the list of labeled samples together with the model can be extended.

We implemented our classification algorithm using *Extra Randomized Trees*. We choose this classifier due to its efficiency over several types of classification problems [6]. However, our framework accepts a wide range of classifiers. Likewise, the system can learn from several types of features. For the purposes of this paper, we limit our analysis to behaviors related to history leaking. In particular, we profile the way in which extensions interact with certain components of the Application Programming Interface (API) exported to extensions by Chrome.

From all API traces that are extracted, we model the way in which consecutive calls are invoked using n-grams. This detection method has been widely explored for the identification of malicious software. As explained in prior work [25, 28], n-grams are particularly useful to model sequences of elements. The number associated to the "n" is the length of each examined sequence; the system receives labeled sequences and uses them to train a classifier in order to recognize from the sequences of an unknown sample to which label the sample should be assigned. In our system

we tried different depths of the n-grams, namely 1, 2, and 6, and the sequences are consecutive API calls invoked by the extension.

4 EVALUATION

In this section, we evaluate the performance of Ex-Ray in finding extensions that leak browser history. We describe our experimental methodology and results.

4.1 Experimental Setting

An overview of our experimental setup is depicted in Figure 4. Next, individual components are discussed in detail.

Extension Containers. As part of our test environment, we created websites that allow scaling the size of a Web client's browsing history without otherwise changing the behavior of websites. We use local Web and DNS servers so that the browser can connect to our website without sending information to the public Internet. For each execution, we start the experiment from an empty cache in a Docker container using an instrumented Chromium binary. We exercise each extension four times for five minutes each, capturing all generated network traffic. Capturing traffic on the container level provides a full picture of each extension's network interactions.

To reduce measurement noise, we block traffic to Google update services and CRLsets[3] via DNS configuration. We also disable features such as SafeBrowsing and account synchronization.

Considering the maximum URL length of 2,083 characters, we increase the length by 500 characters between stages. Other than changing the length of URLs used, the pages served to the instrumented browsers do not change between stages. The maximum length of URLs generated by us is below 1,600, leaving sufficient space for trackers that submit URLs as GET parameters. For each execution we open 20 pages; thus, if all URLs were transmitted uncompressed we would expect an increase of 10 KB per stage. We store DNS information to group IP traffic by hostname.

Extension Dataset. We crawled the Chrome Web Store and downloaded extensions with 1,000 or more installations. For our analysis, we only consider extensions that can be loaded without crashing. Examples of extensions that could not be loaded are those with manifest files that cannot be parsed or referencing files that are missing from the packages. We discarded 334 such extensions, which left 10,691 extensions for Ex-Ray to analyze.

To establish baseline ground truth, we searched for different types of tracking extensions. We did not use the extensions mentioned in Appendix A as they were not available in the store at the time of these experiments, and the future behavior of tracker endpoints was unclear.

We mainly relied on two approaches to discover history-leaking extensions:

- **Heuristic search**. We look for suspicious hostnames, keywords in network traffic, and apply heuristics to traffic patterns. Through manual verification we confirmed 100 benign extensions and 53 privacy-violating extensions. The dataset contains different types of samples, including aggregate data collection and delivery over HTTP(S) and HTTP2.
- **Honeypot probe**. We registered extensions interacting with our honeypot and verified 38 as connecting back from the public Internet. Figure 1 shows a map of all incoming connections with respect to the time we exercised the extension with unique URLs in the history. Table 1 shows the most installed five malicious extensions with domains connecting to the honeypot. Connections often appear immediately after running the extension, but we detected deferred crawls as well.

[3] gvt1.com, redirector.gvt1.com, clients1.google.com, clients4.google.com

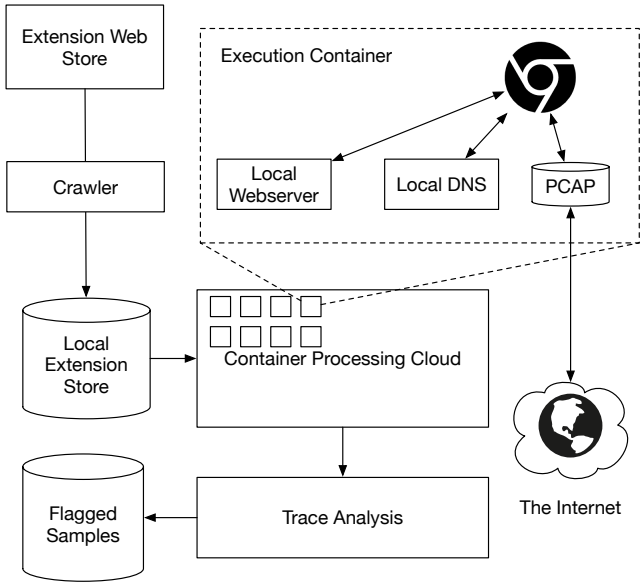


Figure 4: Ex-Ray extension execution overview. After downloading extensions from the Chrome Web Store, we exercise them in containers to collect traces for classification.

We excluded VPN and proxy extensions that redirect traffic via a remote address, as these are not part of our threat model. The connecting clients performed no malicious activities we could identify in our log files. The hostnames of clients that connected to us varied widely. The most popular one was `kontera.com` with 704 connections, followed by AWS endpoints. Interestingly, we received many requests from home broadband connections, such as `*.net-bynet.ru`, often connecting only once. However, we connected four graphs of extensions that were contacted from the same hosts. The biggest graph connects eight extensions with two hosts. The other graphs link five, two, and two extensions. We compiled the data of possible collaboration into Table 3, which is located in the Appendix due to space limitations.

Twelve of these extensions were removed from the Chrome Web Store before our experiments concluded. We did not report these, and we have no indication that the removal could be related to privacy leaks.

Extension Name	installations	Connecting from
Stylish - Custom themes	1,671,326	*.bb.netbynet.ru, *.moscow.rt.ru, *.spb.ertelecom.ru
Pop Up Blocker for Chrome	1,151,178	*.aws.kontera.com, 176.15.177.229, *.bb.netbynet.ru
Desprotetor de Links	251,016	*.aws.kontera.com, *.moscow.rt.ru, *.bb.netbynet.ru
Открытые вкладки (Open Tabs)	97,204	*.dnepro.net, 109.166.71.185, *.k-telecom.org
Similar Sites	45,053	*.aws.kontera.com, *.moscow.rt.ru, *.netbynet.ru

Table 1: Top five extensions connecting to our honeypot with highest installation numbers which are still available in the Chrome Web Store.

4.2 Ex-Ray Results

Tuning of the Unsupervised Component. The first step of Ex-Ray consists of applying linear regression for counterfactual analysis. The linear regression test flags flows if they respect the three parameters explained in Section 3.2. To find the best configuration of these parameters, it is necessary to evaluate the results on a labeled dataset. We use F-Measure as a comparison metric. The strictest policy checks for a minimum of five URLs leaked, a 2% minimum slope, and 90% accuracy. This policy results in an F-Measure of 96.9% and no false positives.

To obtain better results, in our final configuration we use two less strict configurations and flagged as suspicious all flows flagged in both engines. Both configurations check for a minimum of two URLs leaked and 2% minimum slope. However, there is a difference in the last check: while one uses 90% accuracy in checking only the lower bound, the other one uses 80% accuracy checking both the upper and lower bound. As such, the first and last checks are less strict, but the F-Measure does not decrease even if a larger area of the feature space can be flagged. The system correctly flags more flows as with the stricter configuration, but the flows belong to the same extensions already flagged by the previous system.

Labeling Performance. Ex-Ray flagged 212 extensions out of 10,691 as history-leaking using linear regression on the traffic sent by extensions. By checking manually, we noticed that not all the flagged extensions were leaking history. Out of 212 samples, 184 did leak, two were benign, and 26 we could not confirm as clearly benign. It has not been possible to determine if among those 26 extensions there were ones leaking or not. Therefore, to provide a conservative evaluation, we consider Ex-Ray to have 28 benign extensions wrongly identified as history-leaking.

As mentioned earlier, detection systems can be prone to false negatives. To measure this for our system, we spot-checked a representative sample of extensions reported as benign. To establish baseline false negatives we scanned our pcap files for leaks and reimplemented another system used for brute-force searching extension traffic for obfuscated strings with a fixed set of algorithms [24]. This system flagged 367 extensions which we used for our dataset. The false negative samples we subjected to examination numbered 178. These results lead to a precision of 87%, a recall value equal to 50.13%, and an F1-Measure value equal to 63.66%. The overall accuracy value is 98.03%. These values are reached using only the first step of Ex-Ray that is a completely unsupervised algorithm. As we show below, these results are further improved by the next phases of our system.

Among the extensions flagged by Ex-Ray, there are some interesting case studies (such as the Web of Trust extension) which are discussed thoroughly in Section 5.1.

Prioritizing Extensions. Extensions processed in the previous step are then ranked using the score function defined in Section 3.3. Ideally, a triage system should prioritize extensions that are more likely to be privacy-violating than others. This way, the analyst can invest most of their efforts on specimens that are likely to be worth exploring. Other than the ranking of extensions, Ex-Ray provides a report with an informative breakdown of the contribution of each network flow to the overall score. This is also useful to the analyst for further investigation. We next show snippets of a triage report for three extensions that are ranked high, medium, and low:

```
cicimfkkbejhggfjaabggafffgdnjgjp
    4e+18  connectionstrenth.com
   394.88  a.pnamic.com
    28.22  eluxer.net
     4.48  rules.similardeals.net
     1.16  code.jquery.com

pmmbokildidpgafchfmebmhpoeiganhj
    89.22   static-opt1.kizi.com
    89.22   cdn-opt0.kizi.com
    89.29   cdn-opt1.kizi.com
     6.12   tpc.googlesyndication.com
     3.15   securepubads.g.doubleclick.net

pogchimbndbckepmhaagnapfmlfgnala
     1.00   www.gstatic.com
     1.00   chromium-i18n.appspot.com
     1.00   ssl.gstatic.com
     1.00   localhost
     0.67   www.google.com
```

The snippet first displays the unique identifier of the extension, followed by the score given to each of the network flows used to allegedly leak the history. As mentioned before, we group network flows by hostname using DNS information captured during the execution. When considering the thresholds introduced in Equation 3, the recommendation given by our triage system for these extensions is `likely-leaking`, `possibly-leaking`, and `not-leaking` respectively.

As mentioned, these recommendations are manually verified. The analysis starts with review of permissions and source code,

looking for access to `chrome.tabs`, `chrome.webRequest` interception, and other methods of history access. Next, an analyst checks for elements inserted into the DOM that leak the referrer. Finally, requests generated by background scripts and other recorded network traffic is checked.

To quantify the performance of our triage system, we first study how it ranks extensions given by the unsupervised system with respect to the baseline ground truth described above (see Section 4.1). 73 extensions are flagged as `likely-leaking`. Out of those, all but one were manually verified to leak (99%).

Next, the analyst is tasked with verifying 121 additional extensions from the `possibly-leaking` category. Out of those, only one is confirmed to be benign, 24 are marked as inconclusive, and the rest (80%) are confirmed to leak. When ordering the triage score from the bottom, it is easy to find extensions that behave legitimately. For the purpose of this paper, the analyst vetted approximately 100 extensions as `not-leaking`.

We emphasize that the purpose of this phase is not to exhaustively label all leaking extensions, nor to obtain a comprehensive understanding of non-leaking extensions. Instead, the aim is to obtain a slice of those extensions where the quality of the ground truth is enough to apply supervised learning. As a byproduct of this manual verification, we have gained a number of insights into the ecosystem of unwanted extensions which is discussed later in the Section 5.

Classification Results. Our last experiment evaluates the effectiveness of the supervised system introduced in Section 3.4. We aim at understanding the performance of Ex-Ray in classifying leaking extensions using API call traces.

We rely on the dataset labeled and vetted in previous stages of our experiment. We split the dataset between training and testing set using a k-fold cross-validation approach, which has been widely applied in the past [19].

To collect behavioral data from extensions, we implemented a Clang LibTooling program that instruments Chromium's source code. In particular, we instrumented the following components of the Chromium framework: *extensions*, *chrome/browser/extensions*, and *chrome/browser/extensions*.[4]

To evaluate the results, we refer to precision (or positive predictive value) and recall (or sensitivity). With reference to detecting leaking extensions, we judge the performance by the F1-score, as it represents the harmonic mean of precision and recall. For the sake of completeness we also report the proportion of correct predictions (accuracy).

Table 2 shows results over a five-fold split using random sampling. Classification results indicate that we can accurately identify when extensions are leaking by examining their behavior. After evaluating different sizes of n-grams, we obtained our best results with $n = 2$ at 96.43% F1 followed by $n = 6$. When looking at the histogram of APIs executed by the extension ($n = 1$), the performance drops about 7%.

Among the most informative features in our best setting, we can observe calls to different API packages related to the manipulation of URLs such as *extensions.common.url_pattern*, as well as the manipulation of runtime code (JavaScript) associated with the preferences of an extension. In particular, the following two API calls are predominantly seen together in history-leaking extensions: *extensions.browser.extension_prefs.GetExtensionPref()* → *chrome.browser.extensions.shared_user_script_master.GetScriptsMetadata()*.

Type	Prec.	Recall	ACC	F1
n-gram=1	87.36%	98.19%	87.30%	92.46%
n-gram=2	93.56%	99.49%	94.14%	**96.43%**
n-gram=6	92.18%	99.23%	92.70%	95.58%

Table 2: *n*-gram classification results for varying *n*.

5 DISCUSSION

In this section, we describe and discuss a number of findings resulting from this work. First, we present two new types of tracking behavior found using our system. We then present the most prevalent types of trackers and discuss their fundamental differences together with the issues due to invasive tracking. Finally, we discuss evasion strategies.

5.1 High Profile Leaks

We cover two particularly relevant cases. These pose new challenges as they are immune to state of the art privacy leak detection systems. Both were discovered through unsupervised detection by Ex-Ray.

WOT: Web of Trust, Website Reputation Ratings. Web of Trust (WOT) is a widely used extension with 1.2M installations. The extension gives users a ranking of trustworthiness of visited websites. WOT came under scrutiny in March 2016 when it was found to be selling browsing data.[5] A feature that distinguishes WOT from other extensions is usage of strong encryption at the extension level. It comes with a cryptographic library (`crypto.js`) that encrypts tracking payloads with RC4 on top of transfer via HTTPS, thus hiding contents from data leakage analysis systems that can interpose on and inspect HTTPS content. This extension was automatically flagged by Ex-Ray with a triage score of 61,598 (outstanding, given that > 1 is considered suspicious) and is undetectable by currently available systems that rely on string matching and can be evaded by the use of encryption [24].

The permissions requested by WOT are access to all sites, the ability to modify requests, and access to all tabs. The library will track every visited website, including websites on internal networks. POST data or keystrokes are not monitored, however.

While the history leak is part of the advertised functionality, less invasive ways of implementation are possible. Either storing the reputation database in the extension, or only sending the domain portion to the server, as opposed to the whole URL. For example, Google Safe Browsing offers similar functionality via an offline database.

[4]Excluding unit tests files, we inserted 11,132 trace points in 923 files collecting 6,125 function parameters.

[5]https://thehackernews.com/2016/11/web-of-trust-addon.html

CouponMate: Coupon Codes & Deals. CouponMate is a shopping application that offers to help users search for applicable coupons. This extension collects and leaks browsing data via WebSockets, a recently-standardized protocol not analyzed by prior work [24], but one that is rising in popularity. In our dataset we found that 103 (0.96%) of extensions use WebSockets. Unlike previous work, Ex-Ray is oblivious to protocols and flagged this extension with a triage score of 20.1, which is a high alert for a human analyst.

Similarly to Web of Trust, the leaks are part of the main functionality. However, implementing it in a more privacy-aware fashion is possible. For example by adding a button to perform a search for coupons on demand, as opposed to sending every URL.

5.2 Browser-enabled Tracking

Trackers are popular on websites and well-studied. However, they are fundamentally different from tracking in browser extensions. Websites need to opt-in to use a tracker, and their scope is limited to their own website unless purposefully shared. Furthermore, visitors can use tracker-blockers to opt-out of tracking with extensions such as Ghostery. Conversely, in browser extensions the scope of tracking is not limited to a single website, but collects information on all websites the extension has permission to access. Furthermore, no tools exist to reduce the impact on privacy.

Mozilla Firefox. Using a prototype we developed for Firefox extensions, we scanned the most popular available ones in the store. We found five extensions with over 400,000 total installations which were tracking user behavior outside of extensions, and reported them to Mozilla. Out of these, three were removed from the store because they did not disclose tracking in their privacy statement. However, this type of tracking is generally tolerated for Firefox, and as a result we have not further pursued notifications on that platform.

5.3 Foundations Towards Solutions

A combination of these suggested solutions would mitigate the problem of invasive tracking in browser extensions:

- Analyze extensions submitted to extension stores with tools that check for tracking behavior, such as our proposed system Ex-Ray. Users can then be warned that their browsing history will potentially be leaked.
- Implement a new browser extension API to inspect and block traffic to trackers generated by other extensions in background scripts. No such API currently exists. Filtering approaches have proven effective for tracking on websites. This could be used in conjunction with established blacklists (e.g.: EasyList), or further extended with entries generated by systems such as Ex-Ray. An integration into this model could help filter background traffic.
- Consider invasive tracking as a violation of the single purpose rule in extension stores, analogously to ad injection. Such a policy would incentivize authors to prevent leaks themselves.

5.4 Evasion

Malware evading detection systems is a well-explored area and is part of the arms race between attackers and defenders. Examples of this include fingerprinting analysis environments or creating more stealthy programs. While no ultimate solutions exist for these problems, Ex-Ray addresses tracking at a fundamental level.

Another approach would be to lay dormant and only leak at a later point in time. However, we have seen with our honeypot experiments that if leaks are utilized, this often happens immediately. Furthermore, there is an economic incentive on the part of attackers to obtain and monetize leaked history as quickly as possible before its value begins to degrade.

Extensions could also pad sent history to show stable traffic behavior or create noise. However, this would either limit the leakage capacity or be easy to detect from simple checks applied by current defense systems if extensions regularly send large amounts of data to mask leakage.

6 RELATED WORK

As with any Web application, browser extensions are third-party code. However, these programs operate with elevated privileges and have access to powerful APIs that can allow access to all content within the browser. Permission systems allow developers to restrict their programs, but extensions have been shown to over-request permissions, effectively de-sensitivizing users. Heule et al. [9] showed that 71% of the top 500 Chrome extensions use permissions that support leaking private information. They proposed an extension design based on mandatory access control to protect user privacy when browsing.

Previous work on privacy-violating browser extensions has found them to be a prevalent problem. It was shown that official extension store quality checks fail to remove such perpetrators. Blog posts have manually analyzed extensions [3, 27] and Starov et al. studied leaks based on keyword search [24]. In contrast, Ex-Ray does not require searching for particular strings and is oblivious to the protocols used by extensions.

IBEX [8] is a research framework to statically verify access control and data flow policies of extensions. Developers have to author their extensions in high-level type safe languages; .NET and a JavaScript subset are supported. Policies are specified in Datalog and allow for finer-grained control as compared to contemporary permission systems.

Egele et al. [5] used a dynamic taint analysis approach based on the QEMU system emulator to detect spyware in Internet Explorer Browser Helper Objects (BHO). BHOs are classified as malicious if they leak sensitive information on the process level.

Hulk [10] is a system that was used for the first large-scale dynamic analysis of Chrome extensions. The authors introduced the concept of Honeypages. This technique generates Web content tailored to an extension to trigger malicious behavior driven by expectations of the extension.

To monetize extensions, maliciously-inclined authors may add or replace ads in the browser with their own. In 2015, a study found 249 Chrome extensions in the Chrome Web Store injecting unwanted ads [26]. The authors identified two drops in their measurement of ad injection. They correlate to Chrome blocking side

loading of extensions, and introduction of the single purpose rule to the Chrome store.

Contemporary websites use a plethora of third-party services that enable developers to quickly add functionality. As a downside, user privacy suffers, since when websites include content from a remote source the trust a user puts into a website is delegated. Nikiforakis et al. [18] studied these delegations and highlighted how widespread this behavior is. As an example, Google Analytics was included in 68% of the top 10,000 websites.

Third-party tracking on websites has been studied extensively. Browsing on seemingly unrelated sites can be observed by third-party trackers and combined into a comprehensive browsing history. Mayer et al. introduced the FourthParty measurement platform [17], discussing privacy implications, technology, and policy perspectives of third-party tracking. Roesner et al. [20] developed client-side defenses to classify and prevent third-party tracking. Recent work has analyzed the history of Web tracking via the Internet Archive's Wayback Machine [11]. The authors found that tracking has steadily increased since 1996. Tracking on the Web has never been as pervasive as it is now.

Browsers are not the only platform prone to leaks of private data. In PiOS [4], Egele et al. statically analyzed applications from iOS app stores. While only a few applications were identified as leaking private user data, more than half leaked unique phone identifiers that can be used by third parties to profile users. Similarly, AndroidLeaks [7] uses data-flow analysis to evaluate Android applications for leaks of private data; they verified leaks in 2,342 applications. Lever et al. show in a longitudinal study of malware [12] that analysis of network traffic is a key factor to early detection.

In the first step of Ex-Ray, we apply linear regression in order to evaluate causality relations [13]. Counterfactual analysis is a relatively simple, but powerful model which has been used in malware traffic analysis before [16]. The authors focused on distinguishing when a certain kind of malware sample acted differently from usual behavior due to certain user activity (triggers). As the work presented noise and some mislabeled conversations, the authors applied Bayesian Inference to assess causality between specific user actions and malware families. In our case, the absence of false positives among the extensions that were not leaking avoided the use of statistical methods to determine whether there is a relation of causality between being a browser extension leaking browser history or not. Linear regression [21] is widely used, for instance as a preparatory step before applying machine learning [23], or as an embedded technique as in SVM [22].

7 LIMITATIONS

Ex-Ray's goal is to flag extensions that collect private data such as browsing history and exfiltrate it to third parties. An actor with the goal to collect user data is interested in collecting it in real time, which is supported by the samples we analyzed.

It is possible that extensions only exfiltrate data after waiting for a period longer than our tests. However, this is at odds with economic incentives due to the decreasing value of leaked data over time, and is thus unlikely from the perspective of the malicious actor. Tracking for the purpose of analysis of large-scale user behavior requires timely data on all websites. The scope of our work is identifying wholesale tracking through extensions.

Extensions that are narrow in scope, e.g., that collect data for a specific website, would not be flagged by our system. We consider stealing of private data on a wider scale. To enhance our system, an approach similar to honeypages in Hulk [10] could be used.

Malicious software that only triggers on narrow conditions can be impossible to exercise. For example, authors could assemble code based on environmental parameters unknown to analysts. A famous example is the Gauss malware.[6] This malware will only trigger on computers that have a specific configuration and is otherwise not decryptable. Global efforts to analyze this malware have failed to date.

Knowing the specifics of our tool, malicious developers could apply evasion techniques, for example transferring a constant amount of data per visited website by padding URLs or captured keystrokes. Evasion is a general concern for any detection system and there exist several avenues to address this [14].

8 CONCLUSION

With this paper we introduce new methods of detecting privacy-violating browser extensions independently of their protocol. We use a combination of supervised and unsupervised methods to find features characteristic to tracking in extensions. We present Ex-Ray, a prototype implementation of our approach for the Chrome browser, and find two extensions in the official Chrome Web Store which leak private information in previously undetectable ways. Privacy leaks in browser extensions are in an arms race, with extensions evading known methods of detection of previous work. We suggest that extensions should be both tested more rigorously when admitted to the store, as well as monitored while they execute within browsers.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation (NSF) under grant CNS-1409738, the EPSRC under grant EP/N008448/1, an EPSRC-funded *Future Leaders in Engineering and Physical Sciences* award, and Secure Business Austria.

REFERENCES

[1] S. Arshad, A. Kharraz, and W. Robertson. Identifying extension-based ad injection via fine-grained web content provenance. In *Proceedings of the 19th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, Paris, FR, 2016.

[2] L. F. Cranor, J. Reagle, and M. S. Ackerman. Beyond concern: Understanding net users' attitudes about online privacy. *The Internet upheaval: raising questions, seeking answers in communications policy*, pages 47–70, 2000.

[3] detectify labs. Chrome extensions - aka total absence of privacy. https://labs.detectify.com/2015/11/19/chrome-extensions-aka-total-absence-of-privacy/, 2015.

[4] M. Egele, C. Kruegel, E. Kirda, and G. Vigna. PiOS: Detecting Privacy Leaks in iOS Applications. In *Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, 2011.

[5] M. Egele, C. Kruegel, E. Kirda, H. Yin, and D. X. Song. Dynamic spyware analysis. In *USENIX annual technical conference (ATC)*, 2007.

[6] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems? *The Journal of Machine Learning Research (JMLR)*, 15(1):3133–3181, Jan. 2014.

[6] http://arstechnica.com/security/2013/03/the-worlds-most-mysterious-potentially-destructive-malware-is-not-stuxnet/

[7] C. Gibler, J. Crussell, J. Erickson, and H. Chen. Androidleaks: automatically detecting potential privacy leaks in android applications on a large scale. In *International Conference on Trust and Trustworthy Computing (TRUST)*. Springer, 2012.
[8] A. Guha, M. Fredrikson, B. Livshits, and N. Swamy. Verified security for browser extensions. In *IEEE Symposium on Security and Privacy (Oakland)*, 2011.
[9] S. Heule, D. Rifkin, A. Russo, and D. Stefan. The most dangerous code in the browser. In *USENIX Hot Topics in Operating Systems (HotOS)*, Kartause Ittingen, Switzerland, 2015.
[10] A. Kapravelos, C. Grier, N. Chachra, C. Kruegel, G. Vigna, and V. Paxson. Hulk: Eliciting malicious behavior in browser extensions. In *USENIX Security Symposium*, San Diego, CA, 2014.
[11] A. Lerner, A. K. Simpson, T. Kohno, and F. Roesner. Internet jones and the raiders of the lost trackers: An archaeological study of web tracking from 1996 to 2016. In *USENIX Security Symposium*, Austin, TX, 2016.
[12] C. Lever, P. Kotzias, D. Balzarotti, J. Caballero, and M. Antonakakis. A Lustrum of Malware Network Communication: Evolution and Insights. In *Proceedings of the 38th IEEE Symposium on Security and Privacy*, San Jose, CA, USA, May 2017.
[13] D. Lewis. Counterfactuals and comparative possibility. *Journal of Philosophical Logic*, 2:2161–2173, 1973.
[14] M. Lindorfer, C. Kolbitsch, and P. M. Comparetti. Detecting Environment-Sensitive Malware. In *Recent Advances in Intrusion Detection (RAID)*, 2011.
[15] N. K. Malhotra, S. S. Kim, and J. Agarwal. Internet users' information privacy concerns (iuipc): The construct, the scale, and a causal model. *Information systems research*, 15(4):336–355, 2004.
[16] E. Mariconti, J. Onaolapo, G. Ross, and G. Stringhini. The cause of all evils: Assessing causality between user actions and malware activity. In *USENIX Workshop on Cyber Security Experimentation and Test (CSET)*, 2017.
[17] J. R. Mayer and J. C. Mitchell. Third-party web tracking: Policy and technology. In *IEEE Symposium on Security and Privacy (Oakland)*, 2012.
[18] N. Nikiforakis, L. Invernizzi, A. Kapravelos, S. Van Acker, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. You are what you include: large-scale evaluation of remote javascript inclusions. In *ACM Conference on Computer and Communications Security (CCS)*, 2012.
[19] P. Refaeilzadeh, L. Tang, and H. Liu. Cross-validation. In *Encyclopedia of database systems*, pages 532–538. Springer, 2009.
[20] F. Roesner, T. Kohno, and D. Wetherall. Detecting and defending against third-party tracking on the web. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, San Jose, CA, 2012.
[21] G. A. Seber and A. J. Lee. *Linear regression analysis*. John Wiley & Sons, 2012.
[22] A. J. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3), 2004.
[23] S. Sousa, F. G. Martin, M. C. M. Alvim-Ferraz, and M. C. Pereira. Multiple linear regression and artificial neural networks based on principal components to predict ozone concentrations. *Environmental Modelling & Software*, 22(1):97–103, 2007.
[24] O. Starov and N. Nikiforakis. Extended tracking powers: Measuring the privacy diffusion enabled by browser extensions. In *Proceedings of the 26th International Conference on World Wide Web*, 2017.
[25] K. M. Tan and R. A. Maxion. "why 6?" defining the operational limits of stide, an anomaly-based intrusion detector. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 188–201. IEEE, 2002.
[26] K. Thomas, E. Bursztein, C. Grier, G. Ho, N. Jagpal, A. Kapravelos, D. McCoy, A. Nappa, V. Paxson, P. Pearce, N. Provos, and M. A. Rajab. Ad injection at scale: Assessing deceptive advertisement modifications. In *IEEE Symposium on Security and Privacy (Oakland)*, 2015.
[27] M. Weissbacher. These chrome extensions spy on 8 million users. http://mweissbacher.com/blog/2016/03/31/these-chrome-extensions-spy-on-8-million-users/, 2016.
[28] C. Wressnegger, G. Schwenk, D. Arp, and K. Rieck. A close look on n-grams in intrusion detection: anomaly detection vs. classification. In *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*, pages 67–76. ACM, 2013.
[29] X. Xing, W. Meng, B. Lee, U. Weinsberg, A. Sheth, R. Perdisci, and W. Lee. Understanding malvertising through ad-injecting browser extensions. In *International Conference on World Wide Web (WWW)*. ACM, 2015.

A CASE STUDY OF A LARGE HISTORY DATA COLLECTOR

As case study we look into SimilarWeb, one of the actors in data collection in browser extensions. We conducted this study before developing Ex-Ray, as the findings turned out to be symptomatic for a wider range of extensions, the findings motivated development of our system [27].

SimilarWeb is a company that offers insights into third-party Web analytics. To the end user the functionality is similar to Google Analytics, except that visitors can see traffic details of websites neither they or SimilarWeb are affiliated with. This is useful for analysis of competitors, or to explore new markets for a product.

Using the free version of their service, the presented information includes information on visitors, search, and advertising. The data is detailed, including number of visitors, average visit duration, search keywords used, countries of origin, referring sites, destination sites that the visitors leave through, and others.

A.1 Origins of Data

As the company does not have direct access to these data sources, the displayed data must be extrapolated from data which is accessible to them. This high resolution of data without direct access motivated further investigations. Their website suggest use of four types of data sources including *A panel of monitored devices, currently the largest in the industry.*

A.2 SimilarWeb Chrome Extension

As first step we analyzed the extension offered on their website. The offered main functionality consists of suggesting sites similar to the one currently seen. After reviewing their code and analyzing network traffic, we noticed suspicious behavior. The extension intercepts requests for all websites and reports any URL or search query to SimilarWeb in real time, including metadata such as referrers. We noticed that the JavaScript library used for tracking was developed by another company, Upalytics.[7] The purpose of this library is to track user behavior in Chrome extensions, other platforms are advertised on their website as well, including mobile and desktop. Since this was an external library, we suspected it might be used in other extensions as well.

A.3 Finding More Extensions

After crawling the Chrome extension store we found 42 suspicious extensions by searching for code similarities. To verify malicious behavior we manually analyzed each extension under the aspect of four questions:

- Does the extension have the capability to exfiltrate private data?
- Does tracking happen "out of the box," or does the user have to opt-in?
- Is this behavior mentioned in the terms of service?
- If not, is there a link in the terms of service that explains the behavior of the extension?

All suspicious extensions were able to collect history, all but one were tracking out of the box. The only extension that offered opt-in was *SpeakIt!*, however, they only switched to that model after a user complained about the included spyware on an issue tracker.[8] Of these 42 extensions 19 explain their data collection practices in the terms of service, while 23 do not. Furthermore, out of these 23 extensions 12 have no URL where this would be explained. One

[7] http://www.upalytics.com
[8] https://github.com/skechboy/SpeakIt/issues/12

URL that is used across 13 extensions to explain the privacy ramifications is http://addons-privacy.com. The text is a copy of the `upalytics.com` privacy policy rendered into a PNG image. The content explains that personal information including browsing history and IP data will be collected. Throughout the document instead of specific company names only general language such as "our product" or "company" is used. It can be used as a template for any extension using such tracking. While the URL is shared between extensions, the developers have no obvious connection. Six of the remaining domains point to the same IP address. Some versions of the privacy policy reference California Civil Code Section 1798.83, which allows for inquiry about usage of personal information for direct marketing purposes.[9] We sent emails to two of the email addresses, we received responses after less than a month.

A.4 Network Information

The extensions used nine different hardcoded hostnames to receive tracking information, we found relations linking all 42 extensions. All endpoint domains, `addons-privacy.com`, and `upalytics.com` were registered by Domains by Proxy, a service used to obfuscate ownership of domain names by hiding WHOIS records.[10] All extensions were reporting to subdomains `http://lb.*`. Some of the names of the domains appear to be misleading, suggesting updates or being a searchhelper. Two of the domains (`connectupdate.com`, `secureweb24.net`) were registered 13 seconds apart. Also, the `robots.txt` file used in all cases is the same.

Furthermore, all these IPs belong to the same hoster, XLHost. Eight out of nine of these hosts have all addresses in a /18 network, half of the IPs of the `upalytics.com` endpoint are in another XLHost network. All IPs in use are unique, however, this involves consecutive IP addresses and other neighborhood relationships.

All hosts used round robin DNS, using multiple IPs for each domain name. To examine this closer we compared the distance of IP addresses used by these extensions for tracking. In Figure 5b, the nodes are the nine domain names in use, edges are the grade of distance. By taking into account distances of up to four, we can link together all hostnames used in all 42 extensions. For example: IPs `1.1.1.1` and `1.1.1.3` have a distance of two. As for the labels, the edge between `similarsites.com` and `thetrafficstat.net` reads `6x2`. This means that the domains share six IP addresses with a distance of two. Figure 5a visualizes the distance relationship between `lb.crdui.com` and `lb.datarating.com`.

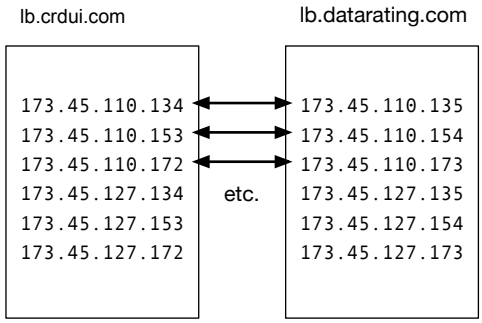
A.5 Reported Extensions

After reporting our findings, all extensions were removed from the Chrome store within 24 hours, including the official SimilarWeb and SimilarSites extensions - a partner site. We hope that loss of installations from the Chrome store will deter developers from bundling malicious libraries in the future.

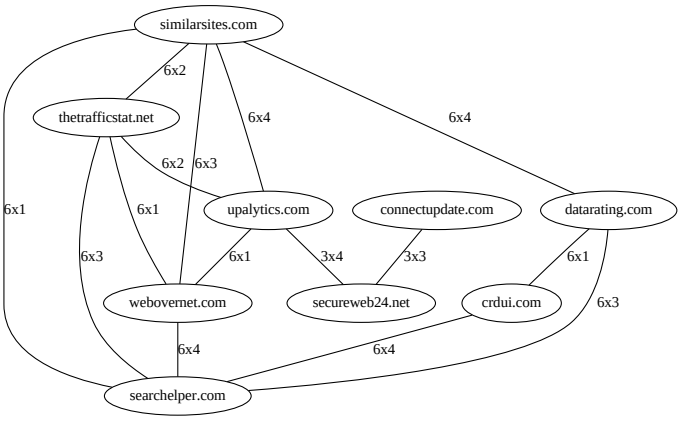
B POSSIBLE COLLABORATION OF TRACKERS

In our honeypot experiment we use URLs unique to each extension. Should we receive incoming connections to such URLs we can link them to the extension which leaked it. As described in Section 4.1 we receive incoming connections often briefly after executing an extension. Other than the behavior over time, another aspect is possible collaboration between extension authors.

When multiple hosts connect to the same generated URL, or hosts connect to multiple URLs unique to an extension, we group them together. These groups are indicators for a possible form of data sharing or shared infrastructure between trackers. An overview of these groups is shown in Table 3.



(a) Neighboring relationships of IPs between seemingly unrelated domains used for monitoring.



(b) Graph linking domain names by IP relationships used in 42 extensions to covertly collect browsing history.

Figure 5: Domains receiving data from the `upalytics.com` library reported to a network of domains that can be linked by IP neighborship.

[9] https://epic.org/privacy/profiling/sb27.html
[10] https://www.domainsbyproxy.com

Extension Name	Connecting from
Sochabra for Stand Alone [translated]	`centro-77.grapeshot.co.uk`
UpTop	`centro-78.grapeshot.co.uk`
500px image downloader	`ec2-176-34-94-65.eu-west-1.compute.amazonaws.com`
BazaarHero	`ec2-54-195-168-122.eu-west-1.compute.amazonaws.com`
DealBeaver	`ec2-54-246-25-158.eu-west-1.compute.amazonaws.com`
EyeEm Image Downloader	
Facebook Image downloader	
Flickr image downloader	
Image Downloader for Facebook & Instagram	
Pinterest Image downloader	
ABC ad blocking China special edition [translated]	`nat-service.aws.kontera.com`
CTRL-ALT-DEL new tab	
Desprotetor de Links	
Pop up blocker for Chrome	
Similar Sites	
Chistodeti	`199.175.48.183`
Woopages	`static.36.51.9.176.clients.your-server.de`

Table 3: In our honeypot probe we observed hosts that connected to multiple URLs unique to extensions, and conversely URLs that received connections from multiple hosts. These relations are possible indicators for a form of data sharing or shared infrastructure between trackers. Each line in this table consists of such a connected group.